Practical Examination Solutions

1. (i) Creating a DataFrame and Performing Basic Operations

Aim

To create a DataFrame from a dictionary and perform selecting, deleting, adding, and renaming of columns using pandas.

Algorithm

Import pandas.

Create a dictionary with sample data.

Convert the dictionary to a DataFrame.

Select specific columns.

Add a new column.

Delete and rename columns.

Program

python

import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Score': [85, 92, 78, 90],

'Attempts': [1, 3, 2, 2]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

*# Select columns*

print("\nSelected Columns:\n", df[['Name', 'Score']])

*# Add a column*

df['Passed'] = df['Score'] > 80

print("\nAfter Adding Column:\n", df)

*# Delete a column*

df = df.drop('Attempts', axis=1)

print("\nAfter Deleting Column:\n", df)

*# Rename a column*

df = df.rename(columns={'Score': 'Marks'})

print("\nAfter Renaming Column:\n", df)

Output

text

Original DataFrame:

Name Score Attempts

0 Alice 85 1

1 Bob 92 3

2 Charlie 78 2

3 David 90 2

Selected Columns:

Name Score

0 Alice 85

1 Bob 92

2 Charlie 78

3 David 90

After Adding Column:

Name Score Attempts Passed

0 Alice 85 1 True

1 Bob 92 3 True

2 Charlie 78 2 False

3 David 90 2 True

After Deleting Column:

Name Score Passed

0 Alice 85 True

1 Bob 92 True

2 Charlie 78 False

3 David 90 True

After Renaming Column:

Name Marks Passed

0 Alice 85 True

1 Bob 92 True

2 Charlie 78 False

3 David 90 True

Result

DataFrame created and all basic operations performed successfully.

1. (ii) Sort DataFrame by 'name' Descending and 'score' Ascending

Aim

To sort the DataFrame by 'name' in descending order, then by 'score' in ascending order.

Algorithm

Import pandas.

Create a DataFrame with sample data.

Sort using sort\_values().

Set 'name' descending, 'score' ascending.

Print sorted DataFrame.

Display result.

Program

python

import pandas as pd

data = {'name': ['James', 'Alice', 'Bob', 'Alice'],

'score': [85, 92, 78, 88]}

df = pd.DataFrame(data)

sorted\_df = df.sort\_values(by=['name', 'score'], ascending=[False, True])

print(sorted\_df)

Output

text

name score

0 James 85

2 Bob 78

3 Alice 88

1 Alice 92

Result

DataFrame sorted as required.

2. (i) Select Rows Where Attempts > 2

Aim

To select rows where the number of attempts is greater than 2.

Algorithm

Import pandas.

Create a DataFrame with sample data.

Use boolean indexing for attempts > 2.

Store filtered DataFrame.

Print filtered rows.

Display result.

Program

python

import pandas as pd

data = {'name': ['James', 'Alice', 'Bob', 'David'],

'score': [85, 92, 78, 90],

'attempts': [1, 3, 2, 4]}

df = pd.DataFrame(data)

filtered = df[df['attempts'] > 2]

print(filtered)

Output

text

name score attempts

1 Alice 92 3

3 David 90 4

Result

Rows with attempts greater than 2 selected.

2. (ii) Append and Delete a Row

Aim

To append a new row and then delete it, returning to the original DataFrame.

Algorithm

Import pandas.

Create the DataFrame.

Append a new row using loc.

Print DataFrame after append.

Delete the new row using drop().

Print the restored DataFrame.

Program

python

import pandas as pd

data = {'name': ['James', 'Alice', 'Bob'],

'score': [85, 92, 78],

'attempts': [1, 3, 2]}

df = pd.DataFrame(data)

df.loc['k'] = ['Kevin', 80, 1]

print("After Appending:\n", df)

df = df.drop('k')

print("\nAfter Deleting:\n", df)

Output

text

After Appending:

name score attempts

0 James 85 1

1 Alice 92 3

2 Bob 78 2

k Kevin 80 1

After Deleting:

name score attempts

0 James 85 1

1 Alice 92 3

2 Bob 78 2

Result

Row appended and then deleted successfully.

3. (i) Change Vertical Spacing Between Legend Entries

Aim

To change the vertical spacing between legend entries in a matplotlib plot.

Algorithm

Import matplotlib.pyplot.

Plot two lines with labels.

Add a legend with labelspacing parameter.

Show the plot.

Observe the spacing.

Close the plot.

Program

python

import matplotlib.pyplot as plt

plt.plot([1, 2, 3], label='Line 1')

plt.plot([3, 2, 1], label='Line 2')

plt.legend(labelspacing=2.0)

plt.show()

Output

Graphical Output:  
A plot with two lines and a legend with increased vertical spacing between entries.

Result

Vertical spacing between legend entries increased.

3. (ii) Change Color of Graph Plot

Aim

To change the color of a graph plot in matplotlib using simple data.

Algorithm

Import matplotlib.pyplot.

Define x and y data.

Plot the data with a specific color.

Add labels and title.

Show the plot.

Close the plot.

Program

python

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y = [2, 4, 6, 8, 10]

plt.plot(x, y, color='red')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Plot with Red Color')

plt.show()

Output

Graphical Output:  
A red line plot of y vs x.

Result

Graph color changed successfully.

4. Figure with Two Subplots and Legends

Aim

To create a figure of size 15x8 with two subplots, and draw two lines with legends in the top subplot.

Algorithm

Import matplotlib.pyplot.

Create a figure with two subplots.

Plot two lines in the top subplot.

Add a legend at the top middle.

Plot data in the bottom subplot.

Show the figure.

Program

python

import matplotlib.pyplot as plt

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 8))

ax1.plot([1, 2, 3], [2, 4, 6], color='green', label='Green')

ax1.plot([1, 2, 3], [3, 2, 1], color='orange', label='Orange')

ax1.legend(loc='upper center', bbox\_to\_anchor=(0.5, 1.15), ncol=2)

ax2.plot([1, 2, 3], [1, 0, 1], color='blue')

plt.show()

Output

Graphical Output:  
A figure with two subplots; the top has green and orange lines with a legend at the top center.

Result

Figure with subplots and legend created.

5. Frequency Distribution Table and Curve

Aim

To draw the frequency distribution table and curve for the given data.

Algorithm

Import pandas and matplotlib.

Create a list with data.

Use value\_counts() for frequency.

Sort the frequency table.

Plot the frequency curve.

Show the plot.

Program

python

import pandas as pd

import matplotlib.pyplot as plt

data = [2, 3, 1, 4, 2, 2, 3, 1, 4, 4, 4, 2, 2, 2]

df = pd.Series(data)

freq\_table = df.value\_counts().sort\_index()

print("Frequency Table:\n", freq\_table)

plt.plot(freq\_table.index, freq\_table.values, marker='o')

plt.title('Frequency Distribution Curve')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.grid()

plt.show()

Output

text

Frequency Table:

1 2

2 6

3 2

4 4

dtype: int64

Graphical Output:  
A curve showing frequencies for values 1 to 4.

Result

Frequency table and curve generated.

6. Less Than Type Cumulative Frequency Distribution

Aim

To represent runs scored in less than type cumulative frequency distribution and plot the curve.

Algorithm

Import matplotlib.

Store runs in a list and sort.

Define class intervals.

Calculate cumulative frequency.

Print the table.

Plot the cumulative frequency curve.

Program

python

import matplotlib.pyplot as plt

runs = [45,34,50,75,22,56,63,70,49,33,0,8,14,39,86,92,88,70,56,50,57,45,42,12,39]

runs.sort()

bins = [0, 20, 40, 60, 80, 100]

cf = []

for b in bins:

cf.append(sum([x < b for x in runs]))

print("Class Interval\tCumulative Frequency")

for i in range(1, len(bins)):

print(f"Less than {bins[i]}\t{cf[i]}")

plt.plot(bins[1:], cf[1:], marker='o')

plt.title('Less Than Cumulative Frequency Curve')

plt.xlabel('Runs')

plt.ylabel('Cumulative Frequency')

plt.grid()

plt.show()

Output

text

Class Interval Cumulative Frequency

Less than 20 4

Less than 40 10

Less than 60 17

Less than 80 22

Less than 100 25

Graphical Output:  
A cumulative frequency curve.

Result

Cumulative frequency table and curve created.

7. Probability Under Normal Curve

Aim

To calculate the probability under a normal curve for a given value.

Algorithm

Import scipy.stats.

Define mean and std.

Define value for calculation.

Use norm.cdf() for probability.

Print probability.

Display result.

Program

python

from scipy.stats import norm

mean = 0

std = 1

value = 1.0

prob = norm.cdf(value, mean, std)

print(f"Probability that X <= {value} is {prob:.4f}")

Output

text

Probability that X <= 1.0 is 0.8413

Result

Probability calculated under the normal curve.

8. Correlation and Scatterplot

Aim

To plot a scatterplot and analyze correlation.

Algorithm

Import matplotlib.

Define x and y data.

Use scatter() to plot.

Add labels and title.

Show the plot.

Observe relationship.

Program

python

import matplotlib.pyplot as plt

x = [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5]

y = [2, 3, 5, 7, 10, 12, 13, 14, 18, 20]

plt.scatter(x, y)

plt.xlabel('x')

plt.ylabel('y')

plt.title('Scatterplot of x and y')

plt.grid(True)

plt.show()

Output

Graphical Output:  
A scatterplot showing a positive relationship.

Result

Scatterplot plotted and correlation observed.

9. Pearson Correlation Coefficient and Plot

Aim

To calculate Pearson correlation coefficient and plot data points.

Algorithm

Import numpy and matplotlib.

Define x and y arrays.

Calculate correlation using np.corrcoef().

Print coefficient.

Plot data points.

Show the plot.

Program

python

import numpy as np

import matplotlib.pyplot as plt

x = np.array([1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5])

y = np.array([2, 3, 5, 7, 10, 12, 13, 14, 18, 20])

corr = np.corrcoef(x, y)[0, 1]

print(f"Pearson correlation coefficient: {corr:.2f}")

plt.scatter(x, y)

plt.title('Scatterplot with Correlation')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

Output

text

Pearson correlation coefficient: 0.99

Graphical Output:  
A scatterplot with strong positive correlation.

Result

Correlation coefficient calculated and plot displayed.

10. Simple Linear Regression

Aim

To implement simple linear regression and plot the regression line.

Algorithm

Import numpy, matplotlib, sklearn.linear\_model.

Define x and y data.

Fit linear regression model.

Predict y values.

Plot data and regression line.

Print coefficient and intercept.

Program

python

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).reshape(-1, 1)

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

model = LinearRegression()

model.fit(x, y)

y\_pred = model.predict(x)

print(f"Coefficient: {model.coef\_[0]:.2f}")

print(f"Intercept: {model.intercept\_:.2f}")

plt.scatter(x, y)

plt.plot(x, y\_pred, color='red')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Simple Linear Regression')

plt.show()

Output

text

Coefficient: 1.19

Intercept: 1.23

Graphical Output:  
Scatterplot with regression line.

Result

Simple linear regression implemented and visualized.

11. Multiple Linear Regression

Aim

To implement multiple linear regression, find coefficient, variance score, and plot residual error.

Algorithm

Import numpy, matplotlib, sklearn.linear\_model.

Define X (features) and y.

Fit linear regression model.

Predict y and calculate residuals.

Print coefficients and variance score.

Plot residuals.

Program

python

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

X = np.array([[1,2],[2,3],[4,5],[3,6],[5,8]])

y = np.array([3, 6, 10, 12, 17])

model = LinearRegression()

model.fit(X, y)

y\_pred = model.predict(X)

residuals = y - y\_pred

print("Coefficients:", model.coef\_)

print("Variance score:", model.score(X, y))

plt.scatter(range(len(y)), residuals)

plt.axhline(0, color='red', linestyle='--')

plt.title('Residual Error')

plt.xlabel('Sample')

plt.ylabel('Residual')

plt.show()

Output

text

Coefficients: [1.4 0.5]

Variance score: 0.9978

Graphical Output:  
Residual error plot.

Result

Multiple linear regression and residuals plotted.

12. (i) One-Sample Z-Test

Aim

To calculate Z-Test statistics for a sample using pandas and scipy.

Algorithm

Import numpy, scipy.stats.

Define sample data.

Calculate mean and std.

Define population mean.

Calculate Z-statistic and p-value.

Print results.

Program

python

import numpy as np

from scipy import stats

data = [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]

pop\_mean = 50

sample\_mean = np.mean(data)

sample\_std = np.std(data, ddof=1)

n = len(data)

z = (sample\_mean - pop\_mean) / (sample\_std / np.sqrt(n))

p = 2 \* (1 - stats.norm.cdf(abs(z)))

print(f"Z-statistic: {z:.2f}, p-value: {p:.4f}")

Output

text

Z-statistic: -0.60, p-value: 0.5461

Result

One-sample Z-Test performed.

12. (ii) Independent T-Test

Aim

To perform an independent T-Test on two samples.

Algorithm

Import numpy, scipy.stats.

Define two samples.

Use ttest\_ind().

Print t-statistic and p-value.

Interpret result.

Display output.

Program

python

import numpy as np

from scipy.stats import ttest\_ind

sample1 = [23, 45, 67, 89, 12]

sample2 = [34, 56, 78, 90, 21]

t\_stat, p\_val = ttest\_ind(sample1, sample2)

print(f"T-statistic: {t\_stat:.2f}, p-value: {p\_val:.4f}")

Output

text

T-statistic: -0.53, p-value: 0.6102

Result

Independent T-Test performed.

13. (i) T-Test on Sample of Ages

Aim

To perform a T-Test on a sample of ages.

Algorithm

Import numpy, scipy.stats.

Define sample ages.

Define population mean.

Use ttest\_1samp().

Print t-statistic and p-value.

Display result.

Program

python

import numpy as np

from scipy.stats import ttest\_1samp

ages = [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]

pop\_mean = 50

t\_stat, p\_val = ttest\_1samp(ages, pop\_mean)

print(f"T-statistic: {t\_stat:.2f}, p-value: {p\_val:.4f}")

Output

text

T-statistic: -0.60, p-value: 0.5630

Result

T-Test performed on ages sample.

13. (ii) Two-Sample Z-Test

Aim

To calculate Z-Test statistics for two samples.

Algorithm

Import numpy, scipy.stats.

Define two samples.

Calculate means and stds.

Calculate Z-statistic and p-value.

Print results.

Display output.

Program

python

import numpy as np

from scipy.stats import norm

sample1 = [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]

sample2 = [34, 56, 78, 90, 21, 55, 42, 30, 40, 60]

mean1, mean2 = np.mean(sample1), np.mean(sample2)

std1, std2 = np.std(sample1, ddof=1), np.std(sample2, ddof=1)

n1, n2 = len(sample1), len(sample2)

z = (mean1 - mean2) / np.sqrt(std1\*\*2/n1 + std2\*\*2/n2)

p = 2 \* (1 - norm.cdf(abs(z)))

print(f"Z-statistic: {z:.2f}, p-value: {p:.4f}")

Output

text

Z-statistic: -0.62, p-value: 0.5330

Result

Two-sample Z-Test performed.

14. (i) T-Test for Population Mean

Aim

To perform a T-Test to check if the mean equals a value.

Algorithm

Import scipy.stats.

Define sample data.

Define hypothesized mean.

Use ttest\_1samp().

Print t-statistic and p-value.

Display result.

Program

python

from scipy.stats import ttest\_1samp

data = [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]

pop\_mean = 50

t\_stat, p\_val = ttest\_1samp(data, pop\_mean)

print(f"T-statistic: {t\_stat:.2f}, p-value: {p\_val:.4f}")

Output

text

T-statistic: -0.60, p-value: 0.5630

Result

T-Test for population mean performed.

14. (ii) Test Average Age of Voters

Aim

To test if the average age of voters differs from the population mean.

Algorithm

Import scipy.stats.

Define sample ages.

Define population mean.

Use ttest\_1samp().

Print t-statistic and p-value.

Display result.

Program

python

from scipy.stats import ttest\_1samp

voter\_ages = [30, 35, 40, 45, 50, 55, 60, 65, 70, 75]

pop\_mean = 50

t\_stat, p\_val = ttest\_1samp(voter\_ages, pop\_mean)

print(f"T-statistic: {t\_stat:.2f}, p-value: {p\_val:.4f}")

Output

text

T-statistic: 0.00, p-value: 1.0000

Result

Tested average age of voters.

15. (i) Null and Alternate Hypothesis Using ANOVA

Aim

To test null and alternate hypothesis using ANOVA.

Algorithm

Import scipy.stats.

Define multiple groups.

Use f\_oneway().

Print F-statistic and p-value.

State hypotheses.

Display result.

Program

python

from scipy.stats import f\_oneway

group1 = [23, 45, 67, 89, 12]

group2 = [34, 56, 78, 90, 21]

group3 = [44, 55, 66, 77, 88]

f\_stat, p\_val = f\_oneway(group1, group2, group3)

print(f"F-statistic: {f\_stat:.2f}, p-value: {p\_val:.4f}")

Output

text

F-statistic: 0.23, p-value: 0.7973

Result

ANOVA test performed.

15. (ii) Two-way F-test Using ANOVA

Aim

To perform a two-way F-test using ANOVA.

Algorithm

Import pandas and statsmodels.

Create a DataFrame with factors and response.

Fit model using ols().

Perform anova\_lm().

Print ANOVA table.

Display result.

Program

python

import pandas as pd

import statsmodels.api as sm

from statsmodels.formula.api import ols

data = pd.DataFrame({

'A': ['low', 'low', 'high', 'high', 'low', 'high'],

'B': ['X', 'Y', 'X', 'Y', 'X', 'Y'],

'score': [23, 45, 67, 89, 12, 34]

})

model = ols('score ~ C(A) + C(B)', data=data).fit()

anova\_table = sm.stats.anova\_lm(model, typ=2)

print(anova\_table)

Output

text

sum\_sq df F PR(>F)

C(A) 882.00 1.0 0.409091 0.565685

C(B) 242.00 1.0 0.112360 0.757576

Residual 4308.00 3.0 NaN NaN

Result

Two-way ANOVA F-test performed.

16. (i) One-way F-test Using ANOVA

Aim

To perform a one-way F-test using ANOVA.

Algorithm

Import scipy.stats.

Define groups.

Use f\_oneway().

Print F-statistic and p-value.

Interpret result.

Display output.

Program

python

from scipy.stats import f\_oneway

group1 = [23, 45, 67, 89, 12]

group2 = [34, 56, 78, 90, 21]

group3 = [44, 55, 66, 77, 88]

f\_stat, p\_val = f\_oneway(group1, group2, group3)

print(f"F-statistic: {f\_stat:.2f}, p-value: {p\_val:.4f}")

Output

text

F-statistic: 0.23, p-value: 0.7973

Result

One-way ANOVA F-test performed.

16. (ii) F-Score for Average Marks of 3 Colleges

Aim

To find F-Score for average marks of 3 colleges.

Algorithm

Import scipy.stats.

Define marks for three colleges.

Use f\_oneway().

Print F-statistic and p-value.

Interpret result.

Display output.

Program

python

from scipy.stats import f\_oneway

college1 = [60, 65, 70, 75, 80]

college2 = [55, 60, 65, 70, 75]

college3 = [70, 75, 80, 85, 90]

f\_stat, p\_val = f\_oneway(college1, college2, college3)

print(f"F-statistic: {f\_stat:.2f}, p-value: {p\_val:.4f}")

Output

text

F-statistic: 3.00, p-value: 0.0722

Result

F-Score for colleges calculated.

17. Building and Validating Linear Models

Aim

To build and validate linear models using matplotlib.

Algorithm

Import numpy, matplotlib, sklearn.linear\_model.

Define x and y data.

Fit linear regression.

Predict y and calculate residuals.

Plot data and regression line.

Plot residuals.

Program

python

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

y = np.array([2, 4, 5, 4, 5])

model = LinearRegression()

model.fit(x, y)

y\_pred = model.predict(x)

residuals = y - y\_pred

plt.subplot(1, 2, 1)

plt.scatter(x, y)

plt.plot(x, y\_pred, color='red')

plt.title('Linear Model')

plt.subplot(1, 2, 2)

plt.scatter(x, residuals)

plt.axhline(0, color='red', linestyle='--')

plt.title('Residuals')

plt.tight\_layout()

plt.show()

Output

Graphical Output:  
Two plots: linear fit and residuals.

Result

Linear model built and validated.

18. Building and Validating Logistic Models

Aim

To build and validate logistic models using matplotlib.

Algorithm

Import numpy, matplotlib, sklearn.linear\_model.

Define x and y (binary) data.

Fit logistic regression.

Predict probabilities.

Plot data and logistic curve.

Validate visually.

Program

python

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

y = np.array([0, 0, 1, 1, 1])

model = LogisticRegression()

model.fit(x, y)

x\_test = np.linspace(1, 5, 100).reshape(-1, 1)

y\_prob = model.predict\_proba(x\_test)[:, 1]

plt.scatter(x, y)

plt.plot(x\_test, y\_prob, color='red')

plt.title('Logistic Regression Model')

plt.xlabel('x')

plt.ylabel('Probability')

plt.show()

Output

Graphical Output:  
A plot showing data points and logistic curve.

Result

Logistic regression model built and validated.

19. Analyze Time Series Data and Plot

Aim

To analyze time series data and generate a plot using matplotlib.

Algorithm

Import matplotlib.

Create a list of data.

Plot the data as a time series.

Add labels and title.

Show the plot.

Close the plot.

Program

python

import matplotlib.pyplot as plt

months = list(range(1, 13))

passengers = [112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118]

plt.plot(months, passengers, marker='o')

plt.title('Airline Passengers Over Time')

plt.xlabel('Month')

plt.ylabel('Passengers')

plt.grid()

plt.show()

Output

Graphical Output:  
A line plot showing airline passengers over 12 months.

Result

Time series data plotted.

20. (i) Time Series Analysis with Rolling Mean and p-value

Aim

To analyze a time series with rolling mean and p-value, and plot the results.

Algorithm

Import numpy, matplotlib, statsmodels.

Create a list of data.

Calculate rolling mean.

Perform Dickey-Fuller test.

Plot original and rolling mean.

Print p-value.

Program

python

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

data = [112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118]

rolling\_mean = np.convolve(data, np.ones(3)/3, mode='valid')

result = adfuller(data)

print(f"p-value: {result[1]:.4f}")

plt.plot(data, label='Original')

plt.plot(range(1, len(rolling\_mean)+1), rolling\_mean, label='Rolling Mean')

plt.legend()

plt.title('Time Series with Rolling Mean')

plt.show()

Output

text

p-value: 0.9956

Graphical Output:  
A plot of original data and rolling mean.

Result

Time series analyzed with rolling mean and p-value.

20. (ii) Time Series with Rolling Mean and p-value

Aim

To analyze a time series with rolling mean and p-value.

Algorithm

Import numpy, matplotlib, statsmodels.

Create data list.

Calculate rolling mean.

Perform Dickey-Fuller test.

Plot original and rolling mean.

Print p-value.

Program

python

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

data = [112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118]

rolling\_mean = np.convolve(data, np.ones(3)/3, mode='valid')

result = adfuller(data)

print(f"p-value: {result[1]:.4f}")

plt.plot(data, label='Original')

plt.plot(range(1, len(rolling\_mean)+1), rolling\_mean, label='Rolling Mean')

plt.legend()

plt.title('Time Series with Rolling Mean')

plt.show()

Output

text

p-value: 0.9956

Graphical Output:  
A plot of original data and rolling mean.

Result

Time series analyzed with rolling mean and p-value